HCT

In this notebook, we present a HCT survival prediction. In this notebook, compared to my previous starter notebooks we teach 5 new things:

- How to tranform `efs` and `efs_time` into single target with `KaplanMeierFitter`.
- How to train `GPU LightGBM model` with `KaplanMeierFitter` target
- How to train `XGBoost with Survivial:Cox loss`
- How to train `CatBoost with Survival:Cox loss`
- How to ensemble 5 models using `scipy.stats.rankdata()`.

Two Competition Approaches

In this competition, there are two ways to train a Survival Model:

- We can input both `efs` and `efs_time` and train a **model that supports** `survival loss like Cox`.
- Transform `efs` and `efs_time` into a single target proxy for `risk score` and train **any model** with `regression loss like MSE`.

In this notebook, we train 5 models. The first 3 models (XGBoost, CatBoost, LightGBM) use bullet point two. And the next 2 models (XGBoost Cox, CatBoost Cox) use bullet point one. Discussion about this notebook is [here][4] and [here][3].

Since this competition's metric is a ranking metric, we ensemble the 5 predictions by first converting each into ranks using `scipy.stats.rankdata()`. Afterward we created a weighted average from the ranks.

Have Fun! Enjoy!

Previous Notebooks

My previous starter notebooks are:

- XGBoost and CatBoost starter [here][1]
- NN (MLP) starter [here][2]

Pip Install Libraries for Metric

Since internet must be turned off for submission, we pip install from my other notebook [here (https://www.kaggle.com/code/cdeotte/pip-install-lifelines)](https://www.kaggle.com/code/cdeotte/pip-install-lifelines) where I downloaded the WHL files.

In [1]:
```
!pip install /kaggle/input/pip-install-lifelines/autograd-1.7.0-py3-none-any.whl
!pip install /kaggle/input/pip-install-lifelines/autograd-gamma-0.5.0.tar.gz
!pip install /kaggle/input/pip-install-lifelines/interface_meta-1.3.0-py3-none-any.whl
!pip install /kaggle/input/pip-install-lifelines/formulaic-1.0.2-py3-none-any.whl
!pip install /kaggle/input/pip-install-lifelines/lifelines-0.30.0-py3-none-any.whl
```

Show hidden output

Load Train and Test

In [2]:

```
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)

test = pd.read_csv("/kaggle/input/equity-post-HCT-survival-predictions/test.csv")
print("Test shape:", test.shape )

train = pd.read_csv("/kaggle/input/equity-post-HCT-survival-predictions/train.csv")
print("Train shape:",train.shape)
train.head()
```

```
Test shape: (3, 58)
Train shape: (28800, 60)
```

Out[2]:

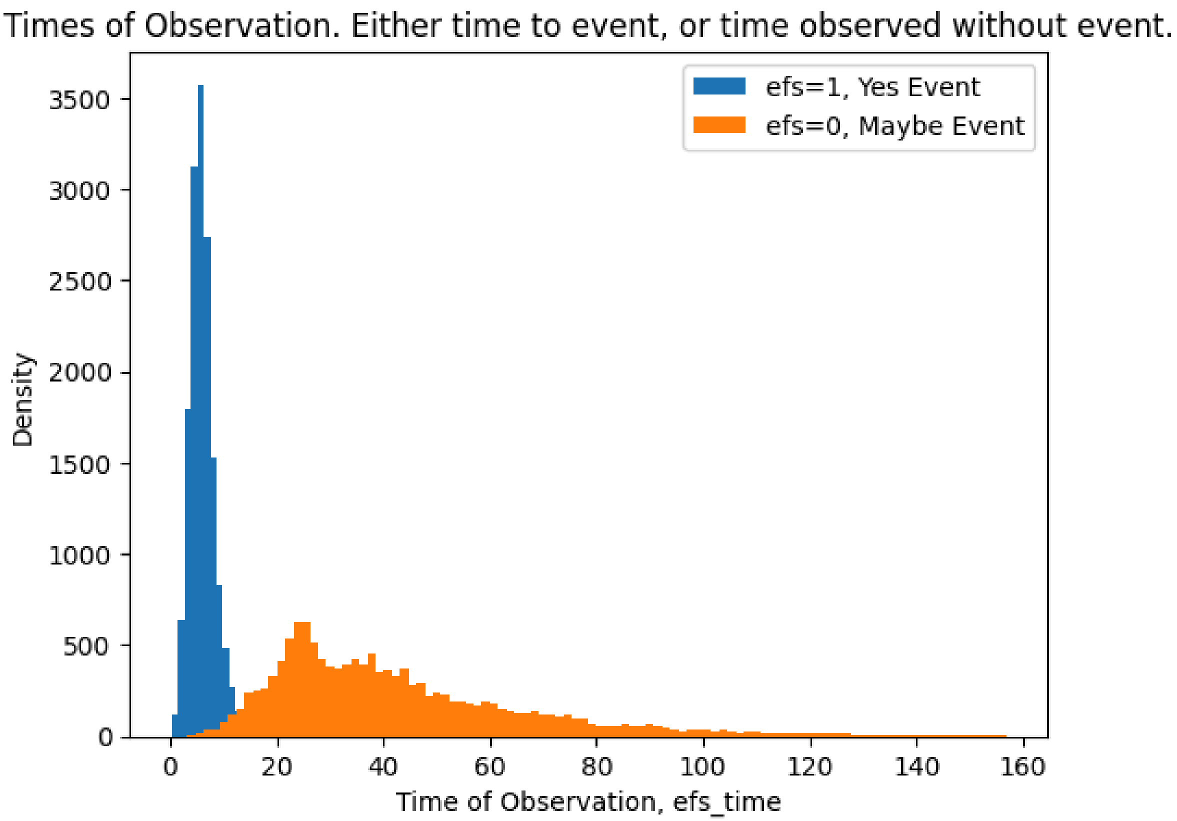
	ID	dri_score	psych_disturb	cyto_score	diabetes	hla_match_c_high	hla_high_res_8
0	0	N/A - non-malignant indication	No	NaN	No	NaN	NaN
1	1	Intermediate	No	Intermediate	No	2.0	8.0
2	2	N/A - non-malignant indication	No	NaN	No	2.0	8.0
3	3	High	No	Intermediate	No	2.0	8.0
4	4	High	No	NaN	No	2.0	8.0

EDA on Train Targets

There are two train targets `efs` and `efs_time`. When `efs==1` we know patient **had an event** and we know time of event is `efs_time`. When `efs==0` we **do not know** if patient had an event or not, but we do know that patient was **without event for at least** `efs_time`.

In [3]:

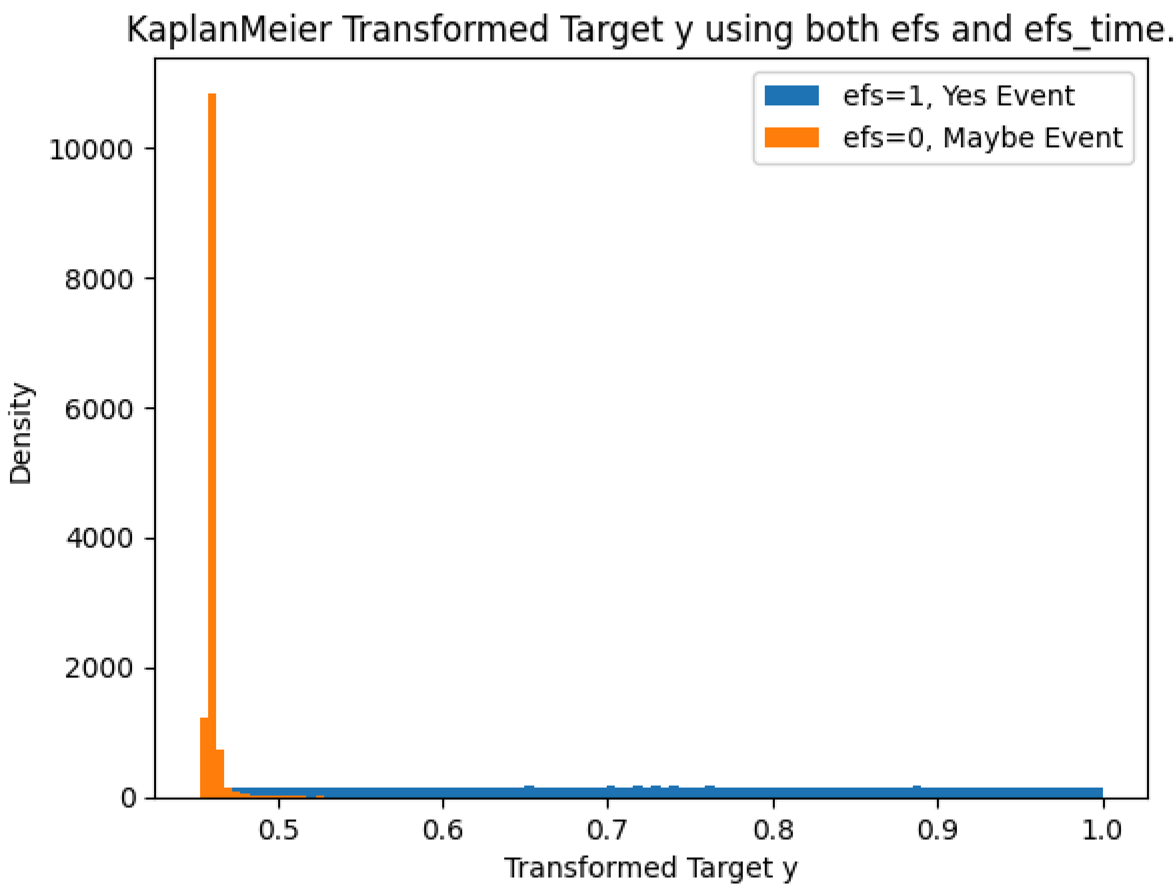
```
plt.hist(train.loc[train.efs==1,"efs_time"],bins=100,label="efs=1, Yes Event")
plt.hist(train.loc[train.efs==0,"efs_time"],bins=100,label="efs=0, Maybe Event")
plt.xlabel("Time of Observation, efs_time")
plt.ylabel("Density")
plt.title("Times of Observation. Either time to event, or time observed without event.")
plt.legend()
plt.show()
```



Transform Two Targets into One Target with KaplanMeier!

Both targets `efs` and `efs_time` provide useful information. We will tranform these two targets into a single target to train our model with. In this competition we need to predict `risk score`. So we will create a target that mimics `risk score` to train our model. (Note this is only one out of many ways to transform two targets into one target. Considering experimenting on your own).

In [4]:
```
from lifelines import KaplanMeierFitter
def transform_survival_probability(df, time_col='efs_time', event_col
='efs'):
    kmf = KaplanMeierFitter()
    kmf.fit(df[time_col], df[event_col])
    y = kmf.survival_function_at_times(df[time_col]).values
    return y
train["y"] = transform_survival_probability(train, time_col='efs_time',
event_col='efs')

plt.hist(train.loc[train.efs==1,"y"],bins=100,label="efs=1, Yes Event")
plt.hist(train.loc[train.efs==0,"y"],bins=100,label="efs=0, Maybe Even
t")
plt.xlabel("Transformed Target y")
plt.ylabel("Density")
plt.title("KaplanMeier Transformed Target y using both efs and efs_tim
e.")
plt.legend()
plt.show()
```



Features

There are a total of 57 features. From these 35 are categorical and 22 are numerical. We will label encode the categorical features. Then our XGB and CAT model will accept these as categorical features and process them special internally. We leave the numerical feature NANs as NANs because GBDT (like XGB and CAT) can handle NAN and will use this information.

In [5]:

```
RMV = ["ID","efs","efs_time","y"]
FEATURES = [c for c in train.columns if not c in RMV]
print(f"There are {len(FEATURES)} FEATURES: {FEATURES}")
```

```
There are 57 FEATURES: ['dri_score', 'psych_disturb', 'cyto_score',
'diabetes', 'hla_match_c_high', 'hla_high_res_8', 'tbi_status', 'ar
rhythmia', 'hla_low_res_6', 'graft_type', 'vent_hist', 'renal_issu
e', 'pulm_severe', 'prim_disease_hct', 'hla_high_res_6', 'cmv_statu
s', 'hla_high_res_10', 'hla_match_dqb1_high', 'tce_imm_match', 'hla
_nmdp_6', 'hla_match_c_low', 'rituximab', 'hla_match_drb1_low', 'hl
a_match_dqb1_low', 'prod_type', 'cyto_score_detail', 'conditioning_
intensity', 'ethnicity', 'year_hct', 'obesity', 'mrd_hct', 'in_vivo
_tcd', 'tce_match', 'hla_match_a_high', 'hepatic_severe', 'donor_ag
e', 'prior_tumor', 'hla_match_b_low', 'peptic_ulcer', 'age_at_hct',
'hla_match_a_low', 'gvhd_proph', 'rheum_issue', 'sex_match', 'hla_m
atch_b_high', 'race_group', 'comorbidity_score', 'karnofsky_score',
'hepatic_mild', 'tce_div_match', 'donor_related', 'melphalan_dose',
'hla_low_res_8', 'cardiac', 'hla_match_drb1_high', 'pulm_moderate',
'hla_low_res_10']
```

In [6]:

```
CATS = []
for c in FEATURES:
    if train[c].dtype=="object":
        CATS.append(c)
        train[c] = train[c].fillna("NAN")
        test[c] = test[c].fillna("NAN")
print(f"In these features, there are {len(CATS)} CATEGORICAL FEATURES: {CATS}")
```

```
In these features, there are 35 CATEGORICAL FEATURES: ['dri_score', 'psych_disturb', 'cyto_score', 'diabetes', 'tbi_status', 'arrhythmia', 'graft_type', 'vent_hist', 'renal_issue', 'pulm_severe', 'prim_disease_hct', 'cmv_status', 'tce_imm_match', 'rituximab', 'prod_type', 'cyto_score_detail', 'conditioning_intensity', 'ethnicity', 'obesity', 'mrd_hct', 'in_vivo_tcd', 'tce_match', 'hepatic_severe', 'prior_tumor', 'peptic_ulcer', 'gvhd_proph', 'rheum_issue', 'sex_match', 'race_group', 'hepatic_mild', 'tce_div_match', 'donor_related', 'melphalan_dose', 'cardiac', 'pulm_moderate']
```

In [7]:
```
combined = pd.concat([train,test],axis=0,ignore_index=True)
#print("Combined data shape:", combined.shape )

# LABEL ENCODE CATEGORICAL FEATURES
print("We LABEL ENCODE the CATEGORICAL FEATURES: ",end="")
for c in FEATURES:

    # LABEL ENCODE CATEGORICAL AND CONVERT TO INT32 CATEGORY
    if c in CATS:
        print(f"{c}, ",end="")
        combined[c],_ = combined[c].factorize()
        combined[c] -= combined[c].min()
        combined[c] = combined[c].astype("int32")
        combined[c] = combined[c].astype("category")

    # REDUCE PRECISION OF NUMERICAL TO 32BIT TO SAVE MEMORY
    else:
        if combined[c].dtype=="float64":
            combined[c] = combined[c].astype("float32")
        if combined[c].dtype=="int64":
            combined[c] = combined[c].astype("int32")

train = combined.iloc[:len(train)].copy()
test = combined.iloc[len(train):].reset_index(drop=True).copy()
```

```
We LABEL ENCODE the CATEGORICAL FEATURES: dri_score, psych_disturb,
cyto_score, diabetes, tbi_status, arrhythmia, graft_type, vent_his
t, renal_issue, pulm_severe, prim_disease_hct, cmv_status, tce_imm_
match, rituximab, prod_type, cyto_score_detail, conditioning_intens
ity, ethnicity, obesity, mrd_hct, in_vivo_tcd, tce_match, hepatic_s
evere, prior_tumor, peptic_ulcer, gvhd_proph, rheum_issue, sex_matc
h, race_group, hepatic_mild, tce_div_match, donor_related, melphala
n_dose, cardiac, pulm_moderate,
```

XGBoost with KaplanMeier

We train XGBoost model for 10 folds and achieve **CV 0.674**!

In [8]:

```
from sklearn.model_selection import KFold
from xgboost import XGBRegressor, XGBClassifier
import xgboost as xgb
print("Using XGBoost version",xgb.__version__)
```

```
Using XGBoost version 2.0.3
```

In [9]:

```
%%time
FOLDS = 10
kf = KFold(n_splits=FOLDS, shuffle=True, random_state=42)

oof_xgb = np.zeros(len(train))
pred_xgb = np.zeros(len(test))

for i, (train_index, test_index) in enumerate(kf.split(train)):

    print("#"*25)
    print(f"### Fold {i+1}")
    print("#"*25)

    x_train = train.loc[train_index,FEATURES].copy()
    y_train = train.loc[train_index,"y"]
    x_valid = train.loc[test_index,FEATURES].copy()
    y_valid = train.loc[test_index,"y"]
    x_test = test[FEATURES].copy()

    model_xgb = XGBRegressor(
        device="cuda",
        max_depth=3,
        colsample_bytree=0.5,
        subsample=0.8,
        n_estimators=2000,
        learning_rate=0.02,
        enable_categorical=True,
        min_child_weight=80,
        #early_stopping_rounds=25,
    )
    model_xgb.fit(
        x_train, y_train,
        eval_set=[(x_valid, y_valid)],
        verbose=500
    )

    # INFER OOF
    oof_xgb[test_index] = model_xgb.predict(x_valid)
    # INFER TEST
    pred_xgb += model_xgb.predict(x_test)
```

```
# COMPUTE AVERAGE TEST PREDS
pred_xgb /= FOLDS
```

```
########################
### Fold 1
########################
[0]     validation_0-rmse:0.17773
[500]   validation_0-rmse:0.15956
[1000]  validation_0-rmse:0.15746
[1500]  validation_0-rmse:0.15650
[1999]  validation_0-rmse:0.15605
########################
### Fold 2
########################
[0]     validation_0-rmse:0.17350
```

```
/opt/conda/lib/python3.10/site-packages/xgboost/core.py:160: UserWa
rning: [05:40:06] WARNING: /workspace/src/common/error_msg.cc:58: F
alling back to prediction using DMatrix due to mismatched devices.
This might lead to higher memory usage and slower performance. XGBo
ost is running on: cuda:0, while the input data is on: cpu.
Potential solutions:
- Use a data structure that matches the device ordinal in the boost
er.
- Set the device for booster before call to inplace_predict.

This warning will only be shown once.

  warnings.warn(smsg, UserWarning)
```

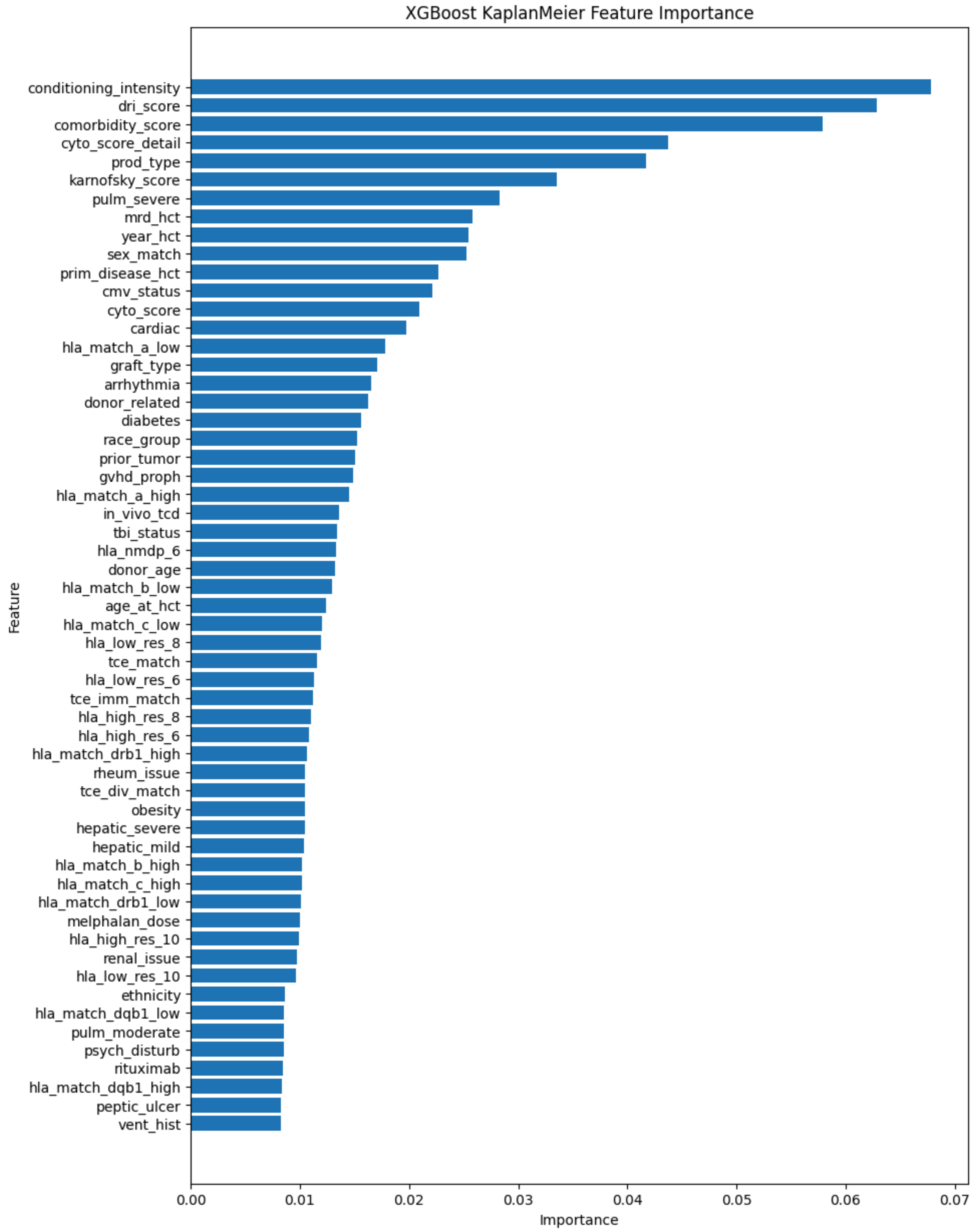
```
[500]   validation_0-rmse:0.15572
[1000]  validation_0-rmse:0.15422
[1500]  validation_0-rmse:0.15356
[1999]  validation_0-rmse:0.15312
#########################
### Fold 3
#########################
[0]     validation_0-rmse:0.17724
[500]   validation_0-rmse:0.15800
[1000]  validation_0-rmse:0.15612
[1500]  validation_0-rmse:0.15538
[1999]  validation_0-rmse:0.15494
#########################
### Fold 4
#########################
[0]     validation_0-rmse:0.17923
[500]   validation_0-rmse:0.16024
[1000]  validation_0-rmse:0.15808
[1500]  validation_0-rmse:0.15713
[1999]  validation_0-rmse:0.15668
#########################
### Fold 5
#########################
[0]     validation_0-rmse:0.17368
[500]   validation_0-rmse:0.15737
[1000]  validation_0-rmse:0.15550
[1500]  validation_0-rmse:0.15474
[1999]  validation_0-rmse:0.15431
#########################
### Fold 6
#########################
[0]     validation_0-rmse:0.17748
[500]   validation_0-rmse:0.15964
[1000]  validation_0-rmse:0.15802
[1500]  validation_0-rmse:0.15738
[1999]  validation_0-rmse:0.15698
#########################
### Fold 7
#########################
[0]     validation_0-rmse:0.17846
[500]   validation_0-rmse:0.16149
[1000]  validation_0-rmse:0.15980
[1500]  validation_0-rmse:0.15907
```

```
[1999]  validation_0-rmse:0.15873
########################
### Fold 8
########################
[0]     validation_0-rmse:0.17457
[500]   validation_0-rmse:0.15782
[1000]  validation_0-rmse:0.15580
[1500]  validation_0-rmse:0.15495
[1999]  validation_0-rmse:0.15453
########################
### Fold 9
########################
[0]     validation_0-rmse:0.17648
[500]   validation_0-rmse:0.16014
[1000]  validation_0-rmse:0.15847
[1500]  validation_0-rmse:0.15762
[1999]  validation_0-rmse:0.15715
########################
### Fold 10
########################
[0]     validation_0-rmse:0.17527
[500]   validation_0-rmse:0.15816
[1000]  validation_0-rmse:0.15628
[1500]  validation_0-rmse:0.15545
[1999]  validation_0-rmse:0.15506
CPU times: user 57.6 s, sys: 383 ms, total: 58 s
Wall time: 52.7 s
```

In [10]:
```
from metric import score

y_true = train[["ID","efs","efs_time","race_group"]].copy()
y_pred = train[["ID"]].copy()
y_pred["prediction"] = oof_xgb
m = score(y_true.copy(), y_pred.copy(), "ID")
print(f"\nOverall CV for XGBoost KaplanMeier =",m)
```

```
/kaggle/usr/lib/eefs-concordance-index/metric.py:59: FutureWarning:
The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain c
urrent behavior or observed=True to adopt the future default and si
lence this warning.
  merged_df_race_dict = dict(merged_df.groupby(['race_group']).grou
ps)


Overall CV for XGBoost KaplanMeier = 0.6737940261928012
```

Show hidden code



CatBoost with KaplanMeier

We train CatBoost model for 10 folds and achieve **CV 0.674**!

In [12]:
```
from catboost import CatBoostRegressor, CatBoostClassifier
import catboost as cb
print("Using CatBoost version",cb.__version__)
```

```
Using CatBoost version 1.2.7
```

In [13]:

```
%%time
FOLDS = 10
kf = KFold(n_splits=FOLDS, shuffle=True, random_state=42)

oof_cat = np.zeros(len(train))
pred_cat = np.zeros(len(test))

for i, (train_index, test_index) in enumerate(kf.split(train)):

    print("#"*25)
    print(f"### Fold {i+1}")
    print("#"*25)

    x_train = train.loc[train_index,FEATURES].copy()
    y_train = train.loc[train_index,"y"]
    x_valid = train.loc[test_index,FEATURES].copy()
    y_valid = train.loc[test_index,"y"]
    x_test = test[FEATURES].copy()

    model_cat = CatBoostRegressor(
        task_type="GPU",
        learning_rate=0.1,
        grow_policy='Lossguide',
        #early_stopping_rounds=25,
    )
    model_cat.fit(x_train,y_train,
              eval_set=(x_valid, y_valid),
              cat_features=CATS,
              verbose=250)

    # INFER OOF
    oof_cat[test_index] = model_cat.predict(x_valid)
    # INFER TEST
    pred_cat += model_cat.predict(x_test)

# COMPUTE AVERAGE TEST PREDS
pred_cat /= FOLDS
```

```
########################
### Fold 1
########################
0:      learn: 0.1743993        test: 0.1760769 best: 0.1760769 (0)
total: 100ms    remaining: 1m 40s
250:    learn: 0.1444360        test: 0.1574516 best: 0.1573758 (24
3)      total: 5.21s    remaining: 15.5s
500:    learn: 0.1364002        test: 0.1570436 best: 0.1569415 (48
6)      total: 10.5s    remaining: 10.4s
750:    learn: 0.1295978        test: 0.1567639 best: 0.1566160 (69
4)      total: 15.5s    remaining: 5.15s
999:    learn: 0.1238415        test: 0.1566584 best: 0.1566160 (69
4)      total: 20s      remaining: 0us
bestTest = 0.1566160111
bestIteration = 694
Shrink model to first 695 iterations.
########################
### Fold 2
########################
0:      learn: 0.1748674        test: 0.1717315 best: 0.1717315 (0)
total: 24ms     remaining: 24s
250:    learn: 0.1445834        test: 0.1532805 best: 0.1532805 (25
0)      total: 4.67s    remaining: 13.9s
500:    learn: 0.1361701        test: 0.1527256 best: 0.1527189 (48
9)      total: 9.01s    remaining: 8.98s
750:    learn: 0.1297626        test: 0.1529458 best: 0.1526560 (55
5)      total: 13.7s    remaining: 4.54s
999:    learn: 0.1241974        test: 0.1531554 best: 0.1526560 (55
5)      total: 20s      remaining: 0us
bestTest = 0.1526560481
bestIteration = 555
Shrink model to first 556 iterations.
########################
### Fold 3
########################
0:      learn: 0.1745178        test: 0.1754648 best: 0.1754648 (0)
total: 20.1ms   remaining: 20.1s
250:    learn: 0.1440720        test: 0.1555157 best: 0.1555074 (24
6)      total: 4.54s    remaining: 13.6s
500:    learn: 0.1358665        test: 0.1552825 best: 0.1552369 (43
0)      total: 9.72s    remaining: 9.68s
750:    learn: 0.1295579        test: 0.1552502 best: 0.1550891 (60
1)      total: 15s      remaining: 4.99s
```

```
999:    learn: 0.1240175        test: 0.1550098 best: 0.1549959 (95
7)      total: 20.4s    remaining: 0us
bestTest = 0.1549959338
bestIteration = 957
Shrink model to first 958 iterations.
#########################
### Fold 4
#########################
0:      learn: 0.1742386        test: 0.1775053 best: 0.1775053 (0)
total: 22ms     remaining: 22s
250:    learn: 0.1443504        test: 0.1572211 best: 0.1571940 (24
9)      total: 4.67s    remaining: 13.9s
500:    learn: 0.1362868        test: 0.1564467 best: 0.1564252 (46
5)      total: 10.1s    remaining: 10.1s
750:    learn: 0.1297536        test: 0.1560188 best: 0.1560188 (75
0)      total: 14.5s    remaining: 4.82s
999:    learn: 0.1242869        test: 0.1560844 best: 0.1559439 (89
7)      total: 19s      remaining: 0us
bestTest = 0.1559438801
bestIteration = 897
Shrink model to first 898 iterations.
#########################
### Fold 5
#########################
0:      learn: 0.1748209        test: 0.1720765 best: 0.1720765 (0)
total: 21.6ms   remaining: 21.6s
250:    learn: 0.1450010        test: 0.1545487 best: 0.1545283 (24
9)      total: 4.06s    remaining: 12.1s
500:    learn: 0.1367038        test: 0.1543137 best: 0.1541771 (37
6)      total: 9.21s    remaining: 9.18s
750:    learn: 0.1301713        test: 0.1545665 best: 0.1541771 (37
6)      total: 14.4s    remaining: 4.78s
999:    learn: 0.1246478        test: 0.1548056 best: 0.1541771 (37
6)      total: 19.5s    remaining: 0us
bestTest = 0.1541770502
bestIteration = 376
Shrink model to first 377 iterations.
#########################
### Fold 6
#########################
0:      learn: 0.1744351        test: 0.1757309 best: 0.1757309 (0)
total: 18.7ms   remaining: 18.7s
250:    learn: 0.1449407        test: 0.1572496 best: 0.1572496 (25
0)      total: 4.04s    remaining: 12s
```

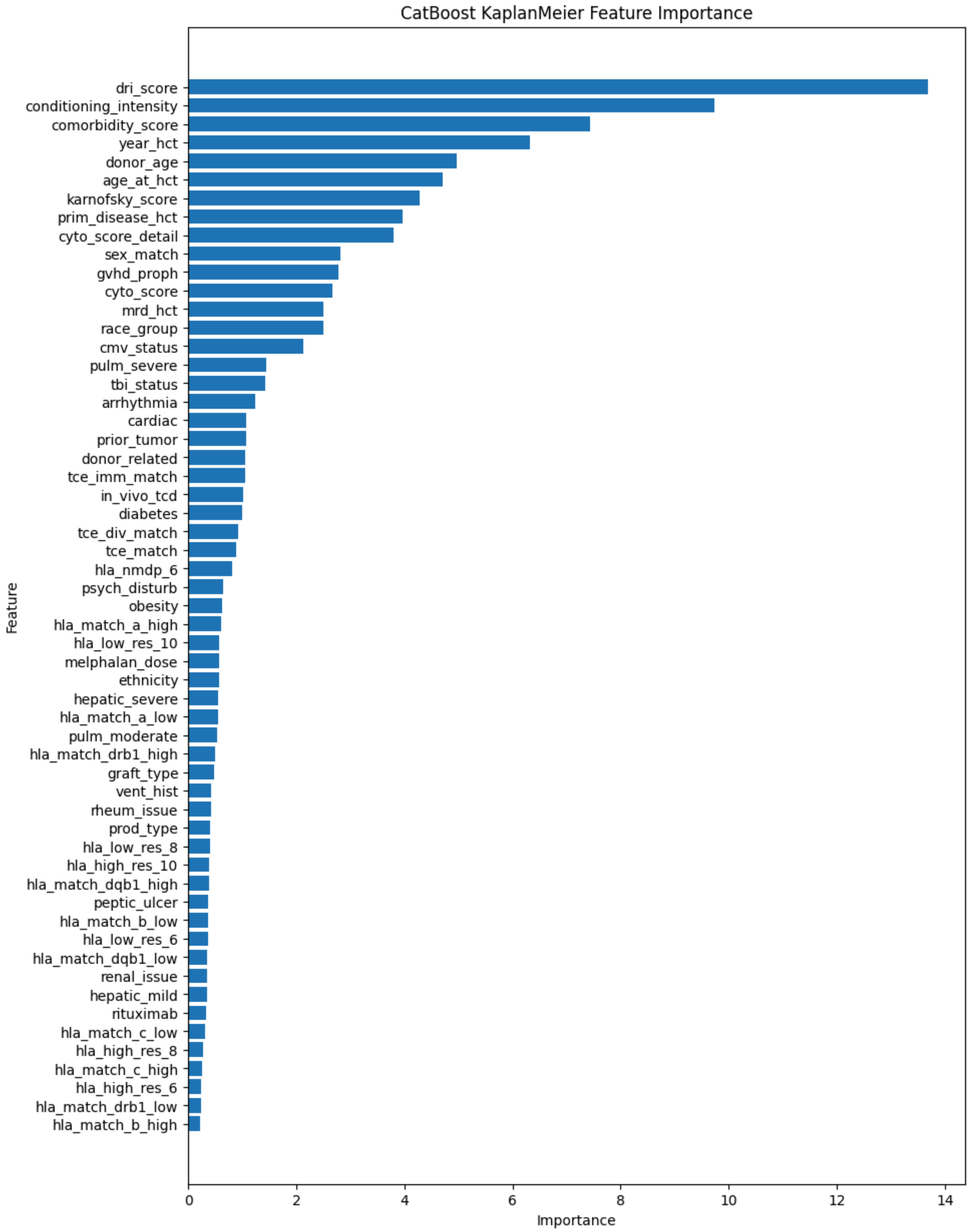
```
500:    learn: 0.1371375        test: 0.1571872 best: 0.1570450 (37
9)      total: 8.59s    remaining: 8.56s
750:    learn: 0.1308869        test: 0.1573460 best: 0.1570450 (37
9)      total: 13.9s    remaining: 4.59s
999:    learn: 0.1252236        test: 0.1579734 best: 0.1570450 (37
9)      total: 19.4s    remaining: 0us
bestTest = 0.1570450034
bestIteration = 379
Shrink model to first 380 iterations.
#########################
### Fold 7
#########################
0:      learn: 0.1743394        test: 0.1767087 best: 0.1767087 (0)
total: 21.4ms   remaining: 21.4s
250:    learn: 0.1446544        test: 0.1592181 best: 0.1592041 (24
7)      total: 4.89s    remaining: 14.6s
500:    learn: 0.1364501        test: 0.1588536 best: 0.1587487 (46
2)      total: 10s      remaining: 10s
750:    learn: 0.1299135        test: 0.1592445 best: 0.1587487 (46
2)      total: 15s      remaining: 4.98s
999:    learn: 0.1244221        test: 0.1593463 best: 0.1587487 (46
2)      total: 19.5s    remaining: 0us
bestTest = 0.1587486709
bestIteration = 462
Shrink model to first 463 iterations.
#########################
### Fold 8
#########################
0:      learn: 0.1746938        test: 0.1728086 best: 0.1728086 (0)
total: 21.4ms   remaining: 21.4s
250:    learn: 0.1442486        test: 0.1553799 best: 0.1553799 (25
0)      total: 4.78s    remaining: 14.3s
500:    learn: 0.1361261        test: 0.1548325 best: 0.1548117 (45
4)      total: 9.83s    remaining: 9.79s
750:    learn: 0.1293713        test: 0.1544145 best: 0.1543675 (73
5)      total: 15.2s    remaining: 5.03s
999:    learn: 0.1236059        test: 0.1545633 best: 0.1543675 (73
5)      total: 20.6s    remaining: 0us
bestTest = 0.154367457
bestIteration = 735
Shrink model to first 736 iterations.
#########################
### Fold 9
#########################
```

```
0:      learn: 0.1744840        test: 0.1748802 best: 0.1748802 (0)
total: 21.9ms   remaining: 21.9s
250:    learn: 0.1442797        test: 0.1583441 best: 0.1583423 (24
5)      total: 5.01s    remaining: 15s
500:    learn: 0.1361857        test: 0.1577813 best: 0.1577649 (49
6)      total: 10.3s    remaining: 10.2s
750:    learn: 0.1297267        test: 0.1573862 best: 0.1573524 (74
7)      total: 15.5s    remaining: 5.15s
999:    learn: 0.1241866        test: 0.1573732 best: 0.1573524 (74
7)      total: 20.7s    remaining: 0us
bestTest = 0.1573523566
bestIteration = 747
Shrink model to first 748 iterations.
#########################
### Fold 10
#########################
0:      learn: 0.1746737        test: 0.1735036 best: 0.1735036 (0)
total: 21.7ms   remaining: 21.7s
250:    learn: 0.1453132        test: 0.1558658 best: 0.1558658 (25
0)      total: 4.98s    remaining: 14.9s
500:    learn: 0.1373193        test: 0.1556390 best: 0.1555556 (35
7)      total: 10.1s    remaining: 10s
750:    learn: 0.1311796        test: 0.1558097 best: 0.1555293 (68
1)      total: 15.6s    remaining: 5.16s
999:    learn: 0.1258469        test: 0.1561605 best: 0.1555293 (68
1)      total: 21s      remaining: 0us
bestTest = 0.1555292758
bestIteration = 681
Shrink model to first 682 iterations.
CPU times: user 8min 27s, sys: 2min 34s, total: 11min 2s
Wall time: 3min 31s
```

In [14]:
```
y_true = train[["ID","efs","efs_time","race_group"]].copy()
y_pred = train[["ID"]].copy()
y_pred["prediction"] = oof_cat
m = score(y_true.copy(), y_pred.copy(), "ID")
print(f"\nOverall CV for CatBoost KaplanMeier =",m)
```

```
/kaggle/usr/lib/eefs-concordance-index/metric.py:59: FutureWarning:
The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain c
urrent behavior or observed=True to adopt the future default and si
lence this warning.
  merged_df_race_dict = dict(merged_df.groupby(['race_group']).grou
ps)


Overall CV for CatBoost KaplanMeier = 0.6740795777257533
```

Show hidden code



LightGBM with KaplanMeier

We train LightGBM model for 10 folds and achieve **CV 0.6725**!

In [16]:
```
from lightgbm import LGBMRegressor
import lightgbm as lgb
print("Using LightGBM version",lgb.__version__)
```

```
Using LightGBM version 4.2.0
```

In [17]:

```
FOLDS = 10
kf = KFold(n_splits=FOLDS, shuffle=True, random_state=42)

oof_lgb = np.zeros(len(train))
pred_lgb = np.zeros(len(test))

for i, (train_index, test_index) in enumerate(kf.split(train)):

    print("#"*25)
    print(f"### Fold {i+1}")
    print("#"*25)

    x_train = train.loc[train_index,FEATURES].copy()
    y_train = train.loc[train_index,"y"]
    x_valid = train.loc[test_index,FEATURES].copy()
    y_valid = train.loc[test_index,"y"]
    x_test = test[FEATURES].copy()

    model_lgb = LGBMRegressor(
        device="gpu",
        max_depth=3,
        colsample_bytree=0.4,
        #subsample=0.9,
        n_estimators=2500,
        learning_rate=0.02,
        objective="regression",
        verbose=-1,
        #early_stopping_rounds=25,
    )
    model_lgb.fit(
        x_train, y_train,
        eval_set=[(x_valid, y_valid)],
    )

    # INFER OOF
    oof_lgb[test_index] = model_lgb.predict(x_valid)
    # INFER TEST
    pred_lgb += model_lgb.predict(x_test)

# COMPUTE AVERAGE TEST PREDS
pred_lgb /= FOLDS
```

```
#########################
### Fold 1
#########################


1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
```

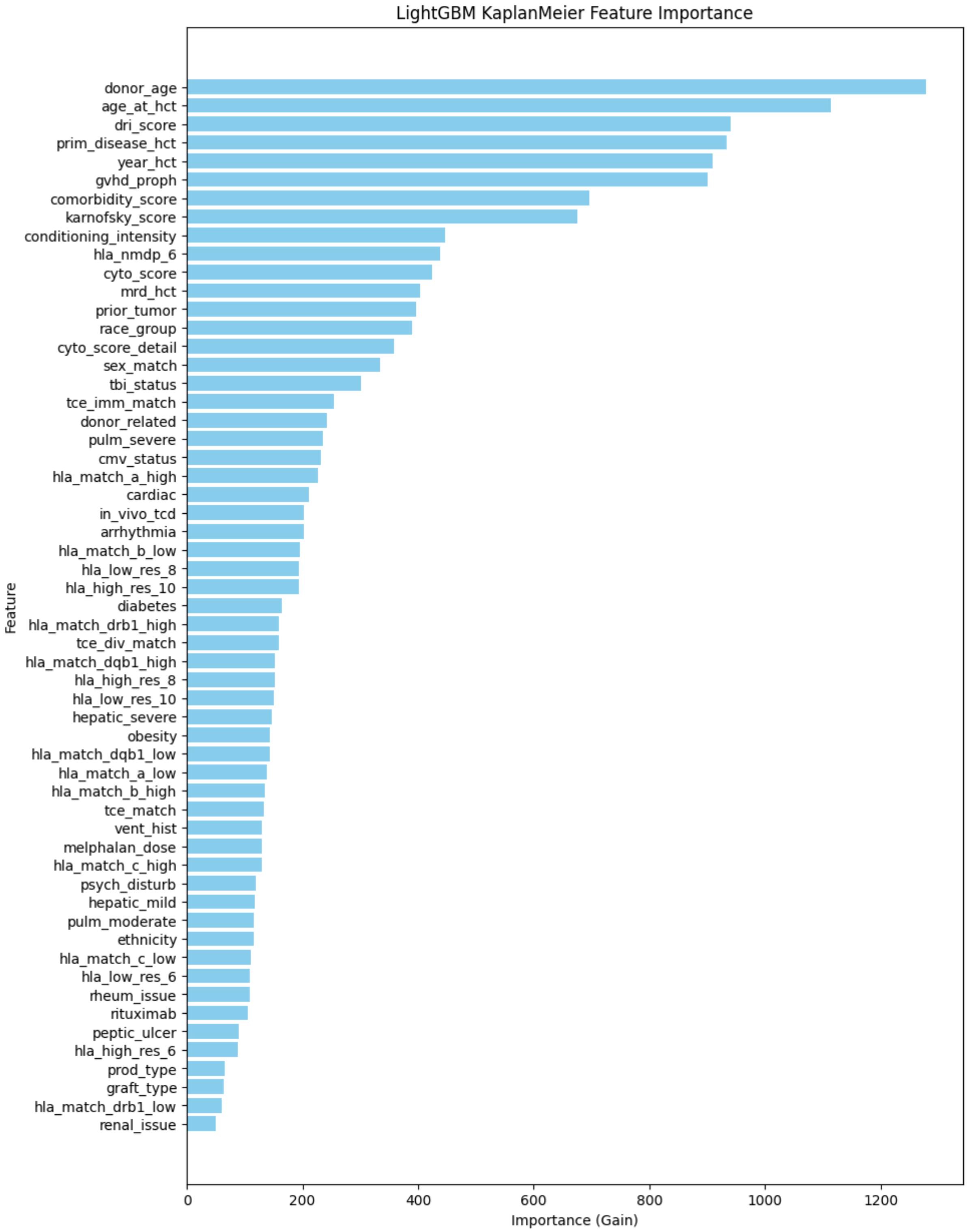
```
#########################
### Fold 2
#########################
#########################
### Fold 3
#########################
#########################
### Fold 4
#########################
#########################
### Fold 5
#########################
#########################
### Fold 6
#########################
#########################
### Fold 7
#########################
#########################
### Fold 8
#########################
#########################
### Fold 9
#########################
#########################
### Fold 10
#########################
```

In [18]:

```
y_true = train[["ID","efs","efs_time","race_group"]].copy()
y_pred = train[["ID"]].copy()
y_pred["prediction"] = oof_lgb
m = score(y_true.copy(), y_pred.copy(), "ID")
print(f"\nOverall CV for LightGBM KaplanMeier =",m)
```

```
/kaggle/usr/lib/eefs-concordance-index/metric.py:59: FutureWarning:
The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain c
urrent behavior or observed=True to adopt the future default and si
lence this warning.
  merged_df_race_dict = dict(merged_df.groupby(['race_group']).grou
ps)


Overall CV for LightGBM KaplanMeier = 0.6725169670618927
```

Show hidden code



XGBoost with Survival:Cox

We train XGBoost using Survival:Cox loss for 10 folds and achieve **CV=672**!

In [20]:
```
# SURVIVAL COX NEEDS THIS TARGET (TO DIGEST EFS AND EFS_TIME)
train["efs_time2"] = train.efs_time.copy()
train.loc[train.efs==0,"efs_time2"] *= -1
```

In [21]:
```
FOLDS = 10
kf = KFold(n_splits=FOLDS, shuffle=True, random_state=42)

oof_xgb_cox = np.zeros(len(train))
pred_xgb_cox = np.zeros(len(test))

for i, (train_index, test_index) in enumerate(kf.split(train)):

    print("#"*25)
    print(f"### Fold {i+1}")
    print("#"*25)

    x_train = train.loc[train_index,FEATURES].copy()
    y_train = train.loc[train_index,"efs_time2"]
    x_valid = train.loc[test_index,FEATURES].copy()
    y_valid = train.loc[test_index,"efs_time2"]
    x_test = test[FEATURES].copy()

    model_xgb_cox = XGBRegressor(
        device="cuda",
        max_depth=3,
        colsample_bytree=0.5,
        subsample=0.8,
        n_estimators=2000,
        learning_rate=0.02,
        enable_categorical=True,
        min_child_weight=80,
        objective='survival:cox',
        eval_metric='cox-nloglik',
    )
    model_xgb_cox.fit(
        x_train, y_train,
        eval_set=[(x_valid, y_valid)],
        verbose=500
    )

    # INFER OOF
    oof_xgb_cox[test_index] = model_xgb_cox.predict(x_valid)
    # INFER TEST
    pred_xgb_cox += model_xgb_cox.predict(x_test)
```

```
# COMPUTE AVERAGE TEST PREDS
pred_xgb_cox /= FOLDS
```

```
#########################
### Fold 1
#########################
[0]     validation_0-cox-nloglik:7.62402
[500]   validation_0-cox-nloglik:7.43513
[1000]  validation_0-cox-nloglik:7.41916
[1500]  validation_0-cox-nloglik:7.41221
[1999]  validation_0-cox-nloglik:7.41085
#########################
### Fold 2
#########################
[0]     validation_0-cox-nloglik:7.61704
[500]   validation_0-cox-nloglik:7.41060
[1000]  validation_0-cox-nloglik:7.39630
[1500]  validation_0-cox-nloglik:7.39059
[1999]  validation_0-cox-nloglik:7.38769
#########################
### Fold 3
#########################
[0]     validation_0-cox-nloglik:7.60952
[500]   validation_0-cox-nloglik:7.40543
[1000]  validation_0-cox-nloglik:7.39056
[1500]  validation_0-cox-nloglik:7.38663
[1999]  validation_0-cox-nloglik:7.38550
#########################
### Fold 4
#########################
[0]     validation_0-cox-nloglik:7.60515
[500]   validation_0-cox-nloglik:7.41071
[1000]  validation_0-cox-nloglik:7.40056
[1500]  validation_0-cox-nloglik:7.39666
[1999]  validation_0-cox-nloglik:7.39720
#########################
### Fold 5
#########################
[0]     validation_0-cox-nloglik:7.62895
[500]   validation_0-cox-nloglik:7.42383
[1000]  validation_0-cox-nloglik:7.40763
[1500]  validation_0-cox-nloglik:7.40042
[1999]  validation_0-cox-nloglik:7.39660
#########################
### Fold 6
#########################
```

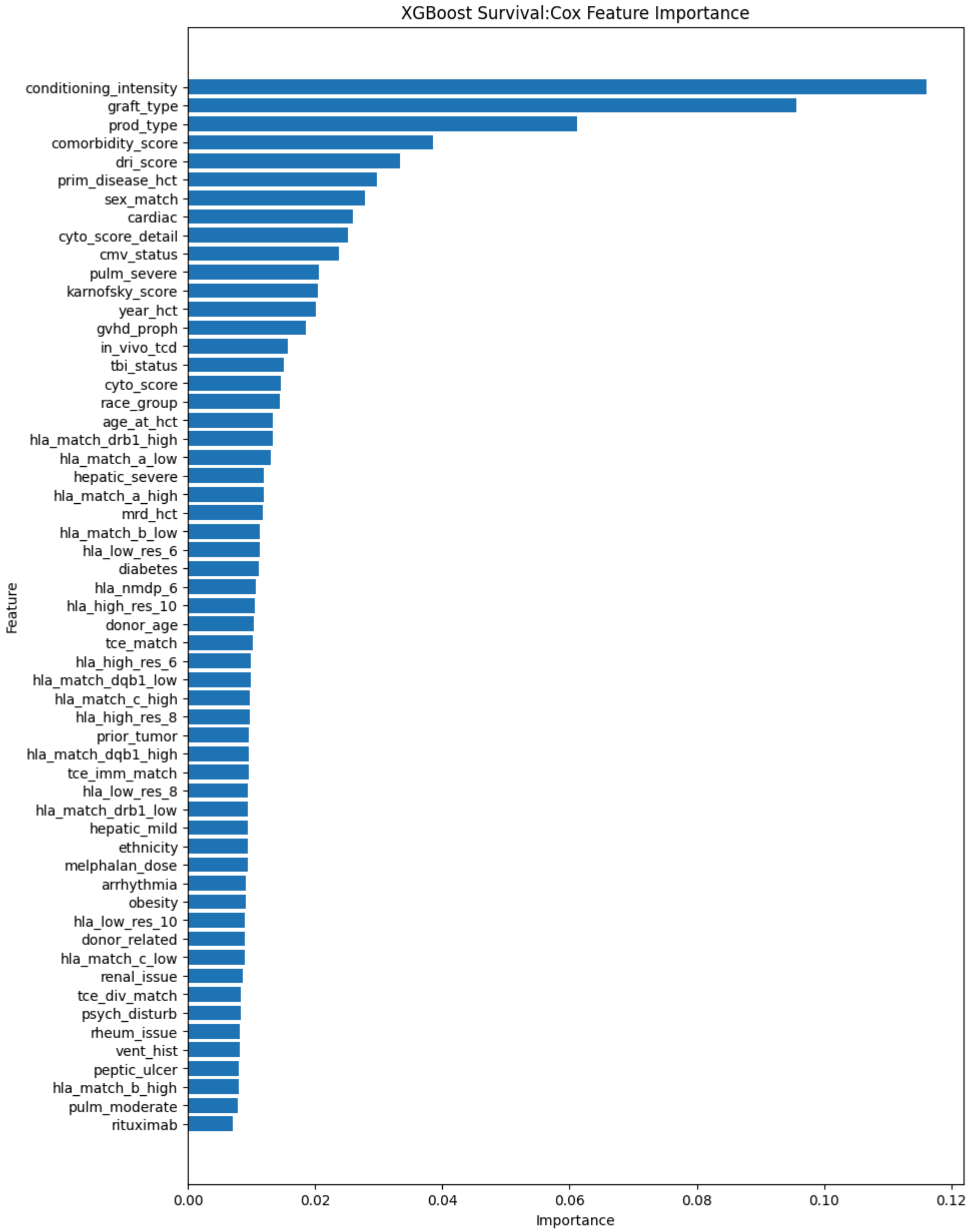
```
[0]     validation_0-cox-nloglik:7.61275
[500]   validation_0-cox-nloglik:7.40221
[1000]  validation_0-cox-nloglik:7.39105
[1500]  validation_0-cox-nloglik:7.38766
[1999]  validation_0-cox-nloglik:7.38749
#########################
### Fold 7
#########################
[0]     validation_0-cox-nloglik:7.62944
[500]   validation_0-cox-nloglik:7.44143
[1000]  validation_0-cox-nloglik:7.42778
[1500]  validation_0-cox-nloglik:7.42296
[1999]  validation_0-cox-nloglik:7.42149
#########################
### Fold 8
#########################
[0]     validation_0-cox-nloglik:7.61662
[500]   validation_0-cox-nloglik:7.44155
[1000]  validation_0-cox-nloglik:7.42844
[1500]  validation_0-cox-nloglik:7.42232
[1999]  validation_0-cox-nloglik:7.41951
#########################
### Fold 9
#########################
[0]     validation_0-cox-nloglik:7.61684
[500]   validation_0-cox-nloglik:7.44634
[1000]  validation_0-cox-nloglik:7.43420
[1500]  validation_0-cox-nloglik:7.42970
[1999]  validation_0-cox-nloglik:7.42799
#########################
### Fold 10
#########################
[0]     validation_0-cox-nloglik:7.61647
[500]   validation_0-cox-nloglik:7.43413
[1000]  validation_0-cox-nloglik:7.42128
[1500]  validation_0-cox-nloglik:7.41845
[1999]  validation_0-cox-nloglik:7.41635
```

In [22]:

```
y_true = train[["ID","efs","efs_time","race_group"]].copy()
y_pred = train[["ID"]].copy()
y_pred["prediction"] = oof_xgb_cox
m = score(y_true.copy(), y_pred.copy(), "ID")
print(f"\nOverall CV for XGBoost Survival:Cox =",m)
```

```
/kaggle/usr/lib/eefs-concordance-index/metric.py:59: FutureWarning:
The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain c
urrent behavior or observed=True to adopt the future default and si
lence this warning.
  merged_df_race_dict = dict(merged_df.groupby(['race_group']).grou
ps)


Overall CV for XGBoost Survival:Cox = 0.6722446470179296
```

Show hidden code



CatBoost with Survival:Cox

We train CatBoost using Survival:Cox loss for 10 folds and achieve **CV=671**!

In [24]:

```
FOLDS = 10
kf = KFold(n_splits=FOLDS, shuffle=True, random_state=42)

oof_cat_cox = np.zeros(len(train))
pred_cat_cox = np.zeros(len(test))

for i, (train_index, test_index) in enumerate(kf.split(train)):

    print("#"*25)
    print(f"### Fold {i+1}")
    print("#"*25)

    x_train = train.loc[train_index,FEATURES].copy()
    y_train = train.loc[train_index,"efs_time2"]
    x_valid = train.loc[test_index,FEATURES].copy()
    y_valid = train.loc[test_index,"efs_time2"]
    x_test = test[FEATURES].copy()

    model_cat_cox = CatBoostRegressor(
        loss_function="Cox",
        #task_type="GPU",
        iterations=400,
        learning_rate=0.1,
        grow_policy='Lossguide',
        use_best_model=False,
    )
    model_cat_cox.fit(x_train,y_train,
              eval_set=(x_valid, y_valid),
              cat_features=CATS,
              verbose=100)

    # INFER OOF
    oof_cat_cox[test_index] = model_cat_cox.predict(x_valid)
    # INFER TEST
    pred_cat_cox += model_cat_cox.predict(x_test)

# COMPUTE AVERAGE TEST PREDS
pred_cat_cox /= FOLDS
```

```
########################
### Fold 1
########################
0:      learn: -137204.2010418  test: -11625.0126498    best: -1162
5.0126498 (0)   total: 70.9ms   remaining: 28.3s
100:    learn: -134245.0940003  test: -11368.0935757    best: -1136
7.7720241 (99)  total: 5.81s    remaining: 17.2s
200:    learn: -133569.4247640  test: -11357.0053940    best: -1135
6.8330165 (182) total: 11.6s    remaining: 11.5s
300:    learn: -133095.7842781  test: -11351.1819262    best: -1135
1.0222775 (299) total: 17.4s    remaining: 5.71s
399:    learn: -132763.5913301  test: -11349.4816640    best: -1134
9.4142821 (327) total: 22.9s    remaining: 0us

bestTest = -11349.41428
bestIteration = 327

########################
### Fold 2
########################
0:      learn: -137014.2912101  test: -11772.8856048    best: -1177
2.8856048 (0)   total: 63.7ms   remaining: 25.4s
100:    learn: -134091.3022715  test: -11485.4489792    best: -1148
5.3225232 (99)  total: 6.23s    remaining: 18.4s
200:    learn: -133312.7852628  test: -11460.6629034    best: -1146
0.6629034 (200) total: 12.1s    remaining: 11.9s
300:    learn: -132843.8300906  test: -11453.5101666    best: -1145
3.1395642 (286) total: 17.9s    remaining: 5.87s
399:    learn: -132444.2041710  test: -11451.6650578    best: -1145
1.1640114 (386) total: 23.6s    remaining: 0us

bestTest = -11451.16401
bestIteration = 386

########################
### Fold 3
########################
0:      learn: -136740.2719659  test: -11983.0664595    best: -1198
3.0664595 (0)   total: 64.8ms   remaining: 25.9s
100:    learn: -133765.3366558  test: -11689.7400344    best: -1168
9.7400344 (100) total: 5.81s    remaining: 17.2s
200:    learn: -133055.1524830  test: -11675.0143694    best: -1167
4.4228636 (194) total: 12.1s    remaining: 11.9s
```

```
300:    learn: -132628.9478783  test: -11670.8603836    best: -1167
0.7024139 (293) total: 17.9s    remaining: 5.88s
399:    learn: -132318.8285745  test: -11674.4124251    best: -1167
0.3801276 (317) total: 23.5s    remaining: 0us

bestTest = -11670.38013
bestIteration = 317

#########################
### Fold 4
#########################
0:      learn: -136474.7243316  test: -12180.0536823    best: -1218
0.0536823 (0)   total: 64.1ms   remaining: 25.6s
100:    learn: -133463.0770737  test: -11892.2690720    best: -1189
2.2690720 (100) total: 5.81s    remaining: 17.2s
200:    learn: -132783.0162317  test: -11878.1443791    best: -1187
7.4964925 (197) total: 11.6s    remaining: 11.5s
300:    learn: -132368.6648703  test: -11875.1126818    best: -1187
4.8392905 (290) total: 17.4s    remaining: 5.72s
399:    learn: -131959.4648801  test: -11873.9173657    best: -1187
3.8885020 (398) total: 23.4s    remaining: 0us

bestTest = -11873.8885
bestIteration = 398

#########################
### Fold 5
#########################
0:      learn: -137321.8175168  test: -11539.7868480    best: -1153
9.7868480 (0)   total: 68.7ms   remaining: 27.4s
100:    learn: -134353.3215180  test: -11253.3822723    best: -1125
3.3822723 (100) total: 5.8s     remaining: 17.2s
200:    learn: -133623.9197023  test: -11235.6006972    best: -1123
5.1441845 (198) total: 11.6s    remaining: 11.5s
300:    learn: -133129.5479754  test: -11236.6138716    best: -1123
3.6543759 (259) total: 17.4s    remaining: 5.72s
399:    learn: -132739.6146843  test: -11237.5755754    best: -1123
3.6543759 (259) total: 23.1s    remaining: 0us

bestTest = -11233.65438
bestIteration = 259

#########################
### Fold 6
```

```
########################
0:      learn: -136830.6351496  test: -11908.3484761    best: -1190
8.3484761 (0)   total: 64ms     remaining: 25.5s
100:    learn: -133900.6412401  test: -11611.3469780    best: -1161
1.3469780 (100) total: 6.18s    remaining: 18.3s
200:    learn: -133046.9692152  test: -11603.0241538    best: -1160
1.2102419 (171) total: 12s      remaining: 11.9s
300:    learn: -132618.4127912  test: -11601.3943921    best: -1159
9.6831955 (279) total: 17.8s    remaining: 5.87s
399:    learn: -132280.1838119  test: -11603.5361790    best: -1159
9.6831955 (279) total: 23.6s    remaining: 0us

bestTest = -11599.6832
bestIteration = 279

########################
### Fold 7
########################
0:      learn: -137331.1086384  test: -11527.8348135    best: -1152
7.8348135 (0)   total: 64.4ms   remaining: 25.7s
100:    learn: -134301.3515021  test: -11269.2842488    best: -1126
9.2842488 (100) total: 5.85s    remaining: 17.3s
200:    learn: -133538.0266711  test: -11258.8797675    best: -1125
8.6810837 (195) total: 11.8s    remaining: 11.7s
300:    learn: -133115.9160399  test: -11261.4263600    best: -1125
8.6810837 (195) total: 18s      remaining: 5.91s
399:    learn: -132757.7938084  test: -11266.3943387    best: -1125
8.6810837 (195) total: 23.6s    remaining: 0us

bestTest = -11258.68108
bestIteration = 195

########################
### Fold 8
########################
0:      learn: -136894.4434350  test: -11865.5004882    best: -1186
5.5004882 (0)   total: 63.8ms   remaining: 25.5s
100:    learn: -133874.3737942  test: -11612.4429632    best: -1161
2.4429632 (100) total: 5.77s    remaining: 17.1s
200:    learn: -133157.3561881  test: -11599.3268855    best: -1159
8.9962380 (191) total: 11.6s    remaining: 11.5s
300:    learn: -132817.2931471  test: -11599.7885483    best: -1159
8.5509036 (220) total: 17.3s    remaining: 5.7s
399:    learn: -132509.2524231  test: -11598.2225128    best: -1159
```

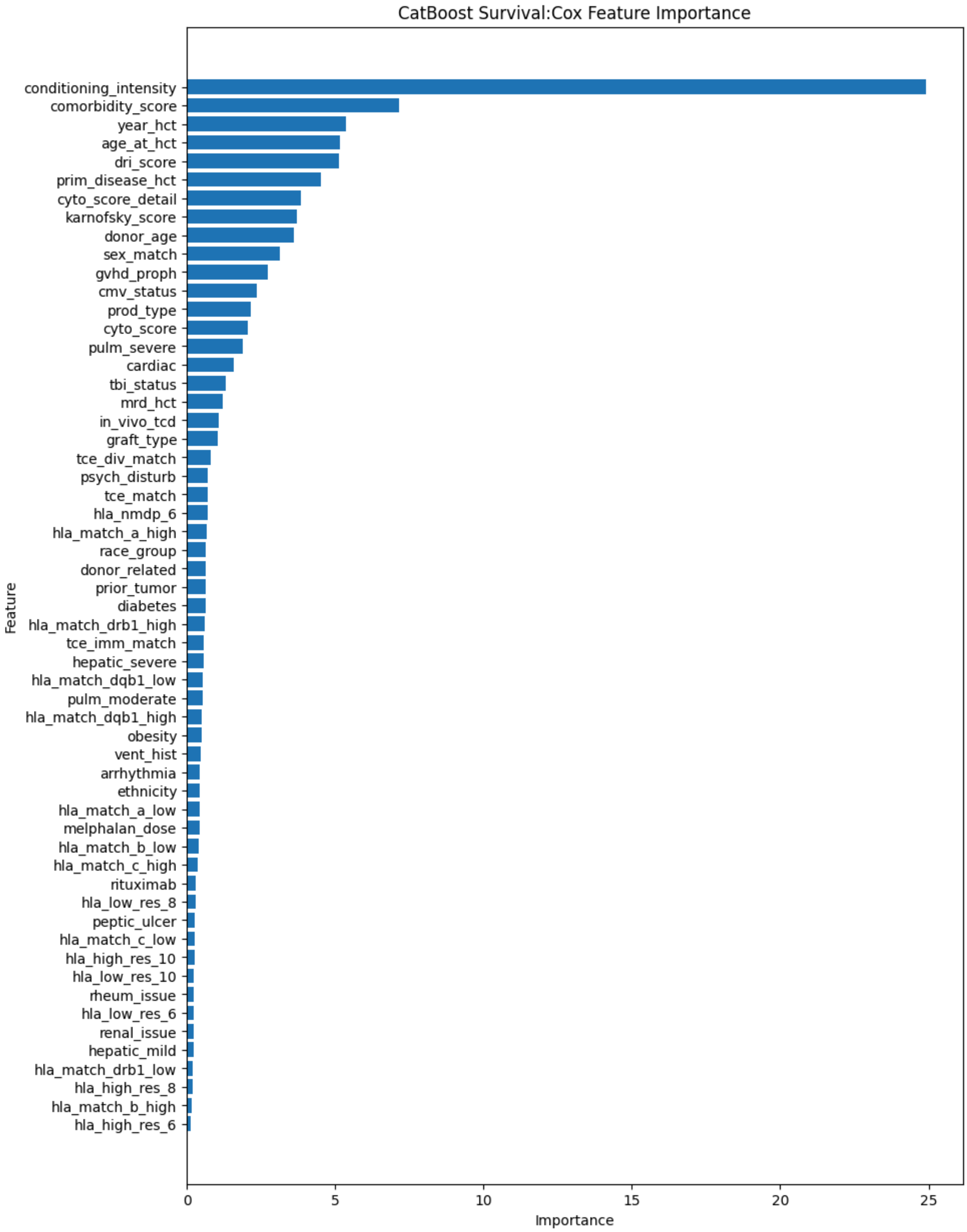
```
5.7294350 (338) total: 23.4s    remaining: 0us

bestTest = -11595.72943
bestIteration = 338

#########################
### Fold 9
#########################
0:      learn: -136897.9544760  test: -11860.2823079    best: -1186
0.2823079 (0)   total: 62.4ms   remaining: 24.9s
100:    learn: -133935.9459575  test: -11622.3451615    best: -1162
2.3451615 (100) total: 5.8s     remaining: 17.2s
200:    learn: -133124.8259221  test: -11615.2248965    best: -1161
3.1851477 (149) total: 11.6s    remaining: 11.5s
300:    learn: -132651.7234484  test: -11615.8087751    best: -1161
3.0622011 (252) total: 17.5s    remaining: 5.75s
399:    learn: -132356.1183940  test: -11616.0147572    best: -1161
3.0622011 (252) total: 23.2s    remaining: 0us

bestTest = -11613.0622
bestIteration = 252

#########################
### Fold 10
#########################
0:      learn: -136968.8520725  test: -11803.6923015    best: -1180
3.6923015 (0)   total: 61.9ms   remaining: 24.7s
100:    learn: -133987.6775754  test: -11539.6778644    best: -1153
9.6778644 (100) total: 6.02s    remaining: 17.8s
200:    learn: -133261.2056712  test: -11531.2596141    best: -1152
9.6792268 (160) total: 12.1s    remaining: 12s
300:    learn: -132741.7177251  test: -11527.6029744    best: -1152
7.5338030 (297) total: 17.9s    remaining: 5.89s
399:    learn: -132387.7444014  test: -11529.0931774    best: -1152
6.9063086 (303) total: 23.6s    remaining: 0us

bestTest = -11526.90631
bestIteration = 303
```

In [25]:

```
y_true = train[["ID","efs","efs_time","race_group"]].copy()
y_pred = train[["ID"]].copy()
y_pred["prediction"] = oof_cat_cox
m = score(y_true.copy(), y_pred.copy(), "ID")
print(f"\nOverall CV for CatBoost Survival:Cox =",m)
```

```
/kaggle/usr/lib/eefs-concordance-index/metric.py:59: FutureWarning:
The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain c
urrent behavior or observed=True to adopt the future default and si
lence this warning.
  merged_df_race_dict = dict(merged_df.groupby(['race_group']).grou
ps)


Overall CV for CatBoost Survival:Cox = 0.6707201406861238
```

Show hidden code



Ensemble CAT and XGB and LGB

We ensemble our XGBoost, CatBoost, LightGBM, XGBoost Cox, and CatBoost Cox using `scipy.stats.rankdata()` and achieve an amazing **CV=0.681** Wow!

In [27]:
```
from scipy.stats import rankdata

y_true = train[["ID","efs","efs_time","race_group"]].copy()
y_pred = train[["ID"]].copy()
y_pred["prediction"] = rankdata(oof_xgb) + rankdata(oof_cat) + rankdata
(oof_lgb)\
                     + rankdata(oof_xgb_cox) + rankdata(oof_cat_cox)
m = score(y_true.copy(), y_pred.copy(), "ID")
print(f"\nOverall CV for Ensemble =",m)
```

```
/kaggle/usr/lib/eefs-concordance-index/metric.py:59: FutureWarning:
The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain c
urrent behavior or observed=True to adopt the future default and si
lence this warning.
  merged_df_race_dict = dict(merged_df.groupby(['race_group']).grou
ps)


Overall CV for Ensemble = 0.680877349730958
```

Create Submission CSV

In [28]:

```
sub = pd.read_csv("/kaggle/input/equity-post-HCT-survival-predictions/sample_submission.csv")
sub.prediction = rankdata(pred_xgb) + rankdata(pred_cat) + rankdata(pred_lgb)\
                     + rankdata(pred_xgb_cox) + rankdata(pred_cat_cox)
sub.to_csv("submission.csv",index=False)
print("Sub shape:",sub.shape)
sub.head()
```

```
Sub shape: (3, 2)
```

Out[28]:

	ID	prediction
0	28800	10.0
1	28801	15.0
2	28802	5.0

In [29]:
```
print(model_xgb)
print(model_xgb_cox)
print(model_cat)
print(model_cat_cox)
print(model_lgb)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.5, device='cuda', early_stopping_ro
unds=None,
             enable_categorical=True, eval_metric=None, feature_typ
es=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.02, max_
bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=80, missing=nan, monotone_constraints
=None,
             multi_strategy=None, n_estimators=2000, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.5, device='cuda', early_stopping_ro
unds=None,
             enable_categorical=True, eval_metric='cox-nloglik',
             feature_types=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.02, max_bin=None, max_cat_threshold=No
ne,
             max_cat_to_onehot=None, max_delta_step=None, max_depth
=3,
             max_leaves=None, min_child_weight=80, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_esti
mators=2000,
             n_jobs=None, num_parallel_tree=None, objective='surviv
al:cox', ...)
<catboost.core.CatBoostRegressor object at 0x7abdd279c820>
<catboost.core.CatBoostRegressor object at 0x7abde0123820>
LGBMRegressor(colsample_bytree=0.4, device='gpu', learning_rate=0.0
2,
              max_depth=3, n_estimators=2500, objective='regressio
n',
              verbose=-1)
```

In [30]:

```
import joblib
import lightgbm as lgb
import json
import os

# Save XGBoost Models
joblib.dump(model_xgb, "/kaggle/working/xgb_model.pkl")
joblib.dump(model_xgb_cox, "/kaggle/working/xgb_cox_model.pkl")
print("✅ XGBoost models saved!")

# Save CatBoost Models
model_cat.save_model("/kaggle/working/cat_model.cbm")
model_cat_cox.save_model("/kaggle/working/cat_cox_model.cbm")
print("✅ CatBoost models saved!")

# Save LightGBM Model
model_lgb.booster_.save_model("/kaggle/working/lgb_model.txt")
print("✅ LightGBM model saved!")

# Save Feature List
with open("/kaggle/working/features.json", "w") as f:
    json.dump(FEATURES, f)
print("✅ Feature list saved!")

# Verify saved files
print("📁 Saved files:", os.listdir("/kaggle/working/"))
```

```
✅ XGBoost models saved!
✅ CatBoost models saved!
✅ LightGBM model saved!
✅ Feature list saved!
📁 Saved files: ['.virtual_documents', 'cat_model.cbm', 'features.j
son', 'submission.csv', 'catboost_info', 'xgb_model.pkl', 'xgb_cox_
model.pkl', 'lgb_model.txt', 'cat_cox_model.cbm']
```

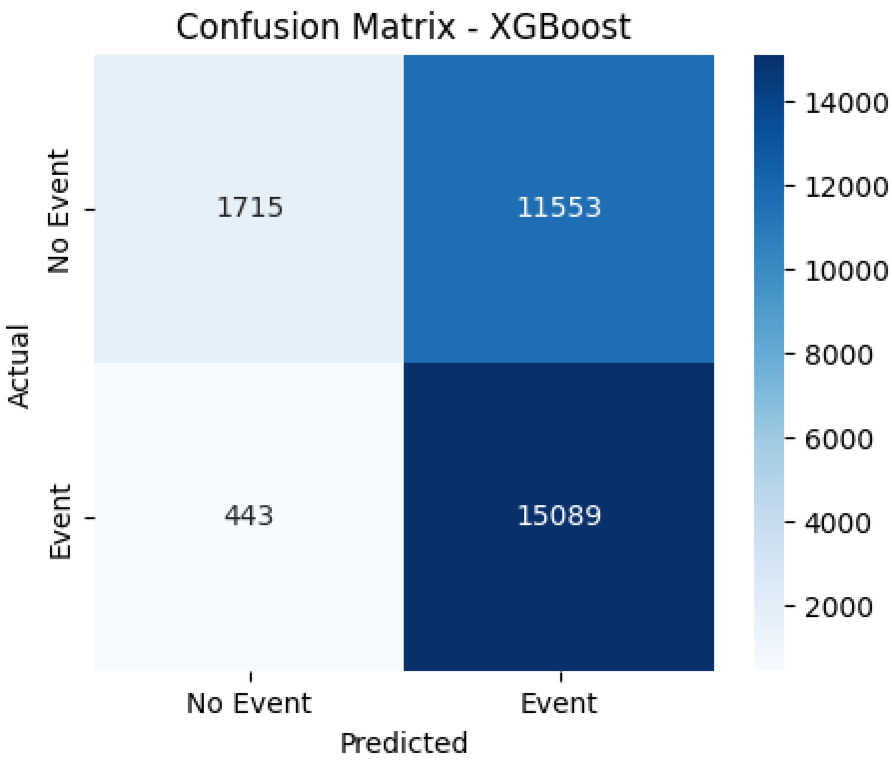
In [31]:
```
!zip -r models.zip /kaggle/working/
```

```
  adding: kaggle/working/ (stored 0%)
  adding: kaggle/working/.virtual_documents/ (stored 0%)
  adding: kaggle/working/cat_model.cbm (deflated 69%)
  adding: kaggle/working/features.json (deflated 59%)
  adding: kaggle/working/submission.csv (deflated 20%)
  adding: kaggle/working/catboost_info/ (stored 0%)
  adding: kaggle/working/catboost_info/learn/ (stored 0%)
  adding: kaggle/working/catboost_info/learn/events.out.tfevents (d
eflated 74%)
  adding: kaggle/working/catboost_info/learn_error.tsv (deflated 5
7%)
  adding: kaggle/working/catboost_info/time_left.tsv (deflated 51%)
  adding: kaggle/working/catboost_info/test/ (stored 0%)
  adding: kaggle/working/catboost_info/test/events.out.tfevents (de
flated 75%)
  adding: kaggle/working/catboost_info/test_error.tsv (deflated 6
4%)
  adding: kaggle/working/catboost_info/tmp/ (stored 0%)
  adding: kaggle/working/catboost_info/catboost_training.json (defl
ated 76%)
  adding: kaggle/working/xgb_model.pkl (deflated 82%)
  adding: kaggle/working/xgb_cox_model.pkl (deflated 81%)
  adding: kaggle/working/lgb_model.txt (deflated 72%)
  adding: kaggle/working/cat_cox_model.cbm (deflated 65%)
```

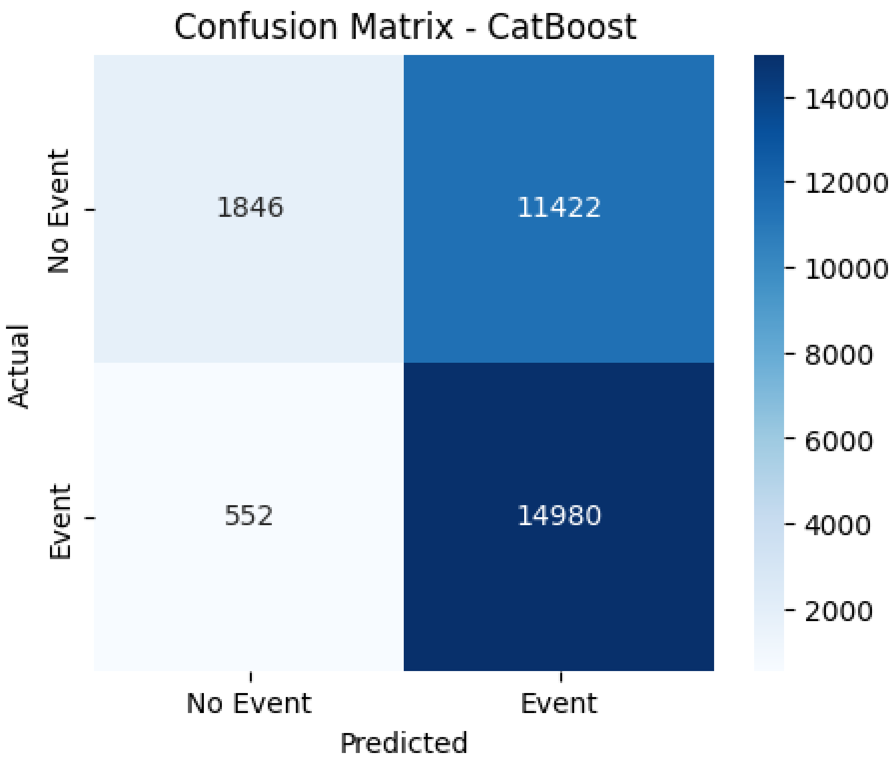
In [32]:

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision
_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

# Define a threshold for binary classification
threshold = 0.5  # Change to np.median(oof_xgb) if needed

# Convert survival scores to binary predictions
y_true = train["efs"]  # Actual event labels

# Convert model outputs into binary labels using the threshold
y_pred_xgb = (oof_xgb > threshold).astype(int)
y_pred_cat = (oof_cat > threshold).astype(int)
y_pred_lgb = (oof_lgb > threshold).astype(int)
y_pred_ensemble = (rankdata(oof_xgb) + rankdata(oof_cat) + rankdata(oof
_lgb) > np.median(rankdata(oof_xgb) + rankdata(oof_cat) + rankdata(oof_
lgb))).astype(int)

# Function to evaluate model performance
def evaluate_model(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    # Display Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No
Event", "Event"], yticklabels=["No Event", "Event"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()

    print(f"🔹 Model: {model_name}")
    print(f"✅ Accuracy: {acc:.4f}")
    print(f"✅ Precision: {prec:.4f}")
    print(f"✅ Recall: {rec:.4f}")
    print(f"✅ F1 Score: {f1:.4f}")
    print("-" * 40)
```

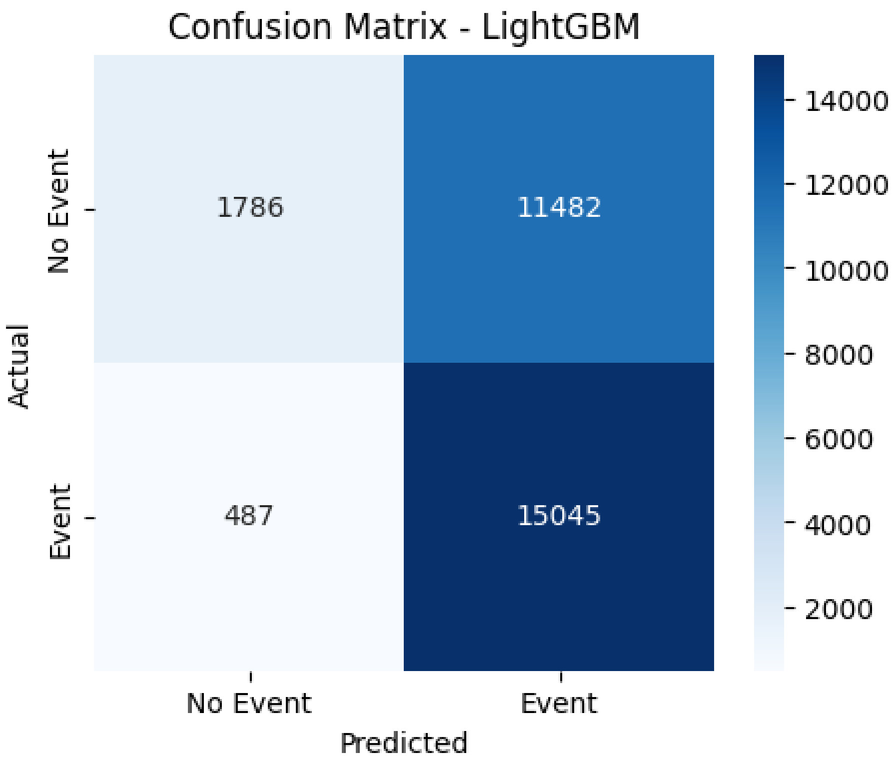
```
# Evaluate all models
evaluate_model(y_true, y_pred_xgb, "XGBoost")
evaluate_model(y_true, y_pred_cat, "CatBoost")
evaluate_model(y_true, y_pred_lgb, "LightGBM")
evaluate_model(y_true, y_pred_ensemble, "Ensemble (XGB + CAT + LGB)")
```



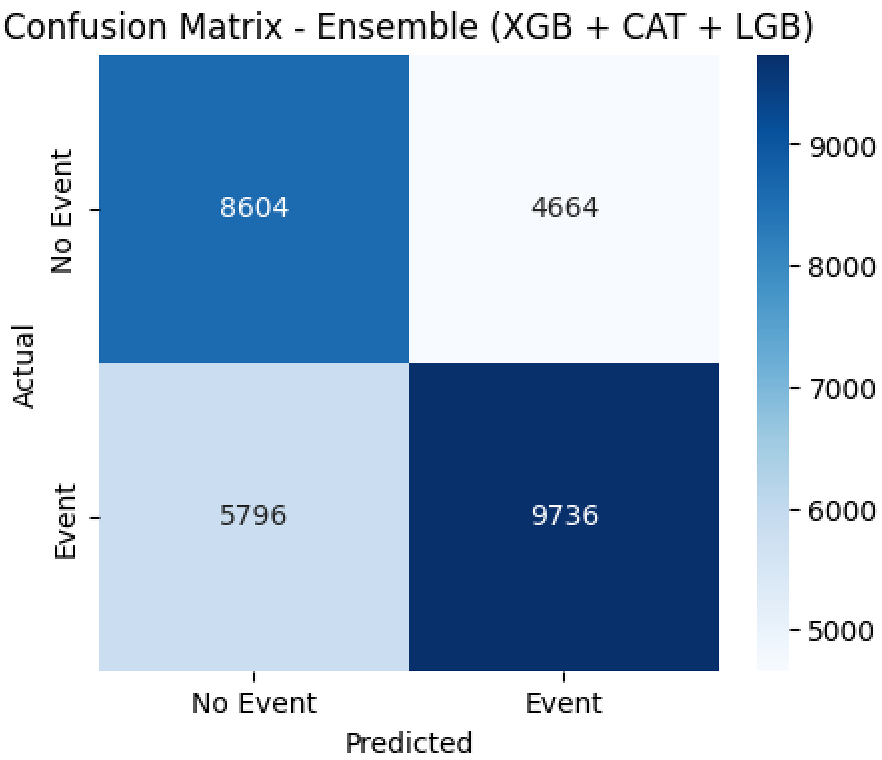
🔹 Model: XGBoost
✅ Accuracy: 0.5835
✅ Precision: 0.5664
✅ Recall: 0.9715
✅ F1 Score: 0.7156




```
🔹 Model: CatBoost
✅ Accuracy: 0.5842
✅ Precision: 0.5674
✅ Recall: 0.9645
✅ F1 Score: 0.7145
----------------------------------------
```



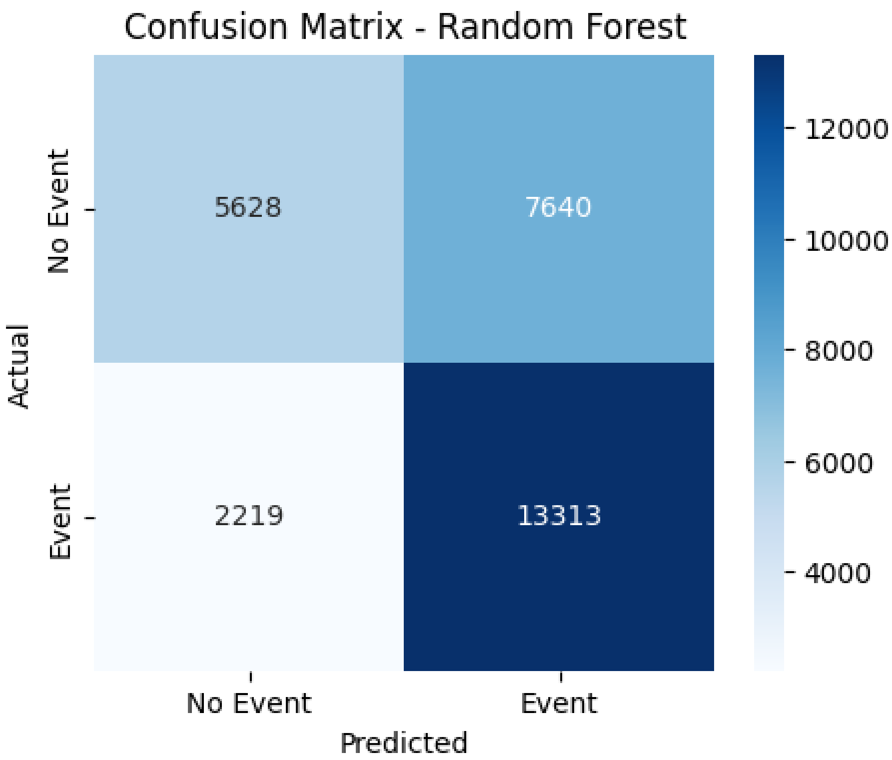

```
🔹 Model: LightGBM
✅ Accuracy: 0.5844
✅ Precision: 0.5672
✅ Recall: 0.9686
✅ F1 Score: 0.7154
----------------------------------------
```




```
🔹 Model: Ensemble (XGB + CAT + LGB)
✅ Accuracy: 0.6368
✅ Precision: 0.6761
✅ Recall: 0.6268
✅ F1 Score: 0.6505
----------------------------------------
```

In [33]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision
_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

# Separate categorical and numerical features
categorical_cols = train[FEATURES].select_dtypes(include=['category',
'object']).columns
numerical_cols = train[FEATURES].select_dtypes(exclude=['category', 'ob
ject']).columns

# Fill missing values
train_filled = train[FEATURES].copy()
train_filled[categorical_cols] = train_filled[categorical_cols].fillna
(train_filled[categorical_cols].mode().iloc[0])
train_filled[numerical_cols] = train_filled[numerical_cols].fillna(trai
n_filled[numerical_cols].median())

# Train Random Forest Classifier
model_rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=5,
    random_state=42,
    n_jobs=-1
)

# Train the model
model_rf.fit(train_filled, train["efs"])

# Predict probabilities and convert to binary classification
oof_rf = model_rf.predict_proba(train_filled)[:, 1]  # Probability of c
lass 1 (event)
y_pred_rf = (oof_rf > 0.5).astype(int)  # Convert to binary classificat
ion

# Function to evaluate model performance
def evaluate_model(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
```

```
    # Display Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Event", "Event"], yticklabels=["No Event", "Event"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()

    print(f"🔹 Model: {model_name}")
    print(f"✅ Accuracy: {acc:.4f}")
    print(f"✅ Precision: {prec:.4f}")
    print(f"✅ Recall: {rec:.4f}")
    print(f"✅ F1 Score: {f1:.4f}")
    print("-" * 40)

# Evaluate Random Forest model
evaluate_model(train["efs"], y_pred_rf, "Random Forest")
```




```
🔹 Model: Random Forest
✅ Accuracy: 0.6577
✅ Precision: 0.6354
✅ Recall: 0.8571
✅ F1 Score: 0.7298
----------------------------------------
```

In [34]:

```
import matplotlib.pyplot as plt
import numpy as np

# Model names
models = ["XGBoost", "CatBoost", "LightGBM", "Ensemble", "Random Fores
t"]

# Metrics for each model
accuracy = [0.5835, 0.5842, 0.5845, 0.6369, 0.6577]
precision = [0.5664, 0.5674, 0.5672, 0.6762, 0.6354]
recall = [0.9715, 0.9645, 0.9686, 0.6269, 0.8571]
f1_score = [0.7156, 0.7145, 0.7154, 0.6506, 0.7298]

# Set bar positions
x = np.arange(len(models))
width = 0.2  # Bar width

plt.figure(figsize=(10, 6))

# Plot bars for each metric
bars1 = plt.bar(x - 1.5*width, accuracy, width, label='Accuracy', color
='blue', alpha=0.7)
bars2 = plt.bar(x - 0.5*width, precision, width, label='Precision', col
or='green', alpha=0.7)
bars3 = plt.bar(x + 0.5*width, recall, width, label='Recall', color='re
d', alpha=0.7)
bars4 = plt.bar(x + 1.5*width, f1_score, width, label='F1 Score', color
='purple', alpha=0.7)

# Function to add labels on top of bars
def add_labels(bars):
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, height + 0.02, f'{hei
ght:.3f}', ha='center', fontsize=10)

# Add values to bars
add_labels(bars1)
add_labels(bars2)
add_labels(bars3)
add_labels(bars4)

# Labels and titles
```

```
plt.xlabel("Models")
plt.ylabel("Metric Scores")
plt.title("Model Performance Comparison")
plt.xticks(x, models)
plt.ylim(0, 1.1)  # Extend y-axis slightly to fit labels

# Add legend
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show the plot
plt.show()
```